Analyze This! 145 Questions for Data Scientists in Software Engineering

Andrew Begel
Microsoft Research
Redmond, WA, USA
Andrew.Begel@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, WA, USA
tzimmer@microsoft.com

ABSTRACT

In this paper, we present the results from two surveys related to data science applied to software engineering. The first survey solicited questions that software engineers would like data scientists to investigate about software, about software processes and practices, and about software engineers. Our analyses resulted in a list of 145 questions grouped into 12 categories. The second survey asked a different pool of software engineers to rate these 145 questions and identify the most important ones to work on first. Respondents favored questions that focus on how customers typically use their applications. We also saw opposition to questions that assess the performance of individual employees or compare them with one another. Our categorization and catalog of 145 questions can help researchers, practitioners, and educators to more easily focus their efforts on topics that are important to the software industry.

Categories and Subject Descriptors: D.2.9 [Management]

General Terms: Management, Human factors, Measurement

Keywords: Data Science, Software Engineering, Analytics

1. INTRODUCTION

Due to the increased availability of data and computing power over the past few years, data science and analytics have become important topics of investigation [1]. Businesses of all types commonly use analytics to better reach and understand their customers [2]. Even sporting teams use analytics to improve their performance as described in the book "Moneyball" [3]. Many software engineering researchers have argued for more use of data for decision-making [4,5,6]. As more and more companies start to analyze their software data, the demand for data scientists in software projects will grow rapidly. Though Harvard Business Review named the job of Data Scientist as the Sexiest Job of the 21st Century [7], by 2018, the U.S. may face a shortage of as many as 190,000 people with analytical expertise and of 1.5 million managers and analysts with the skills to make data-driven decisions, according to a report by the McKinsey Global Institute [8].

Several people have offered advice on the important questions that academic and industry data scientists should focus. In his "Two Solitudes" keynote at the Mining Software Repositories Vision 2020 event in Kingston, Greg Wilson presented a list of ten questions for empirical researchers that a Mozilla developer sent to him in response to the following request: [9]

> *"I'm giving a talk on Monday to a room full of software engineering researchers who are specialists in data-mining software repositories (among other things). If you could get them to tackle any questions at all (well, any related to software or software development), what would you want them to do, and why?"*

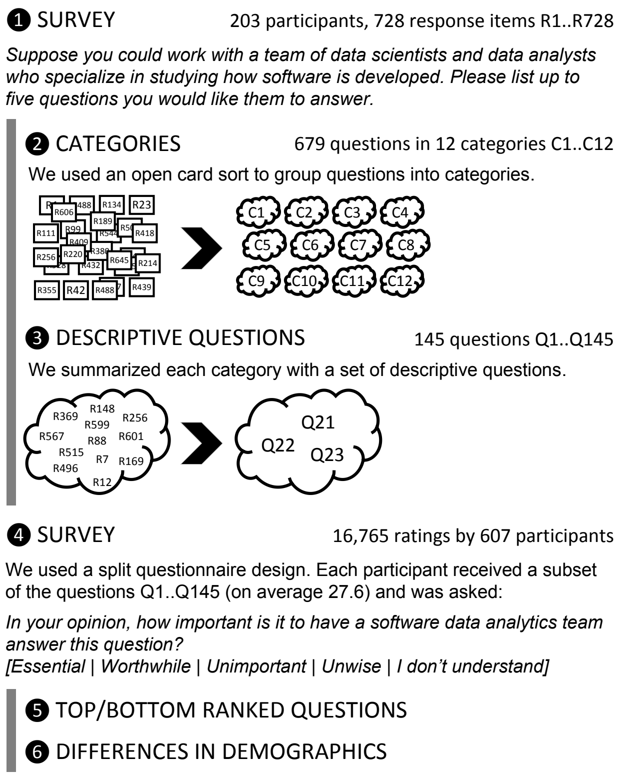
In an introduction to an empirical software engineering panel at ESEC/FSE 2013, Bertrand Meyer emphasized the need for the software engineering community to become more data-driven, and to "shed folkloric advice and anecdotal evidence." He presented a list of 11 questions "crying for evidence" [10] whose answers should be empirical, credible, and useful. By useful, Meyer meant, "providing answers to *questions of interest to practitioners*."

In this paper, we present a ranked list of questions that software engineers want to have answered by data scientists. The list was compiled from two surveys that we deployed among professional software engineers at Microsoft (Section 3).

1. In the first survey, we asked a random sample of 1,500 Microsoft engineers a question similar to Greg Wilson's. We asked *"Please list up to five questions you would like [a team of data scientists who specialize in studying how software is developed] to answer."* After received 728 response items from 203 software engineers, we filtered and grouped them into 679 questions in 12 categories. We then distilled these into 145 descriptive questions (Section 4.1).
2. We deployed a second survey to a new sample of 2,500 Microsoft engineers to help us prioritize the 145 descriptive questions by indicating the most important ones to work on. We received 16,765 ratings from 607 Microsoft engineers. These ratings additionally enabled us to identify differences of opinion between various demographic groups, for example, questions that were more important to testers than to developers (Sections 4.2 and 4.3).

Our findings suggest that engineers favor questions that focus on how customers typically use their applications. We also observe opposition against the use of analytics to assess the performance of individual employees or compare them with one another.

Our catalog of 145 questions is relevant for research, industry practice, and software engineering education (Section 5). For researchers, the descriptive questions outline opportunities to collaborate with industry and influence their software development processes, practices, and tools. For those in industry, the list of questions identifies particular data to collect and analyze to find answers, as well as the need to build collection and analysis tools at industrial scale. Lastly, for educators, the questions provide guidance on what analytical techniques to teach in courses for future data scientists, as well as providing instruction on topics of importance to industry (which students always appreciate). As pointed out earlier [8], there will be huge demand for people who are educated enough to know how to make use of data.



❶ SURVEY 203 participants, 728 response items R1..R728

Suppose you could work with a team of data scientists and data analysts who specialize in studying how software is developed. Please list up to five questions you would like them to answer.

❷ CATEGORIES 679 questions in 12 categories C1..C12

We used an open card sort to group questions into categories.

❸ DESCRIPTIVE QUESTIONS 145 questions Q1..Q145

We summarized each category with a set of descriptive questions.

❹ SURVEY 16,765 ratings by 607 participants

We used a split questionnaire design. Each participant received a subset of the questions Q1..Q145 (on average 27.6) and was asked:

In your opinion, how important is it to have a software data analytics team answer this question?
[Essential | Worthwhile | Unimportant | Unwise | I don't understand]

❺ TOP/BOTTOM RANKED QUESTIONS

❻ DIFFERENCES IN DEMOGRAPHICS

Figure 1. Overview of the research methodology.

In this paper, we make the following contributions:

- We provide a catalog of 145 questions that software engineers would like to ask data scientists about software. (Section 4.1)
- We rank the questions by importance (and opposition) to help researchers, practitioners, and educators focus their efforts on topics of importance to industry. (Sections 4.2 and 4.3)

We issue a call to action to other industry companies and to the academic community. Please replicate our methods and grow the body of knowledge from this modest start. To support replication, we provide the survey instructions and the full list of 145 questions with ratings in a technical report [11].

2. RELATED WORK

The work related to this paper falls into software analytics and (empirical) collections of software engineering questions.

Software Analytics is a subfield of *analytics* with the focus on *software data*. Davenport, Harris, and Morison [2] define analytics "as the use of analysis, data, and systematic reasoning to make decisions." Software data can take many forms such as source code, changes, bug reports, code reviews, execution data, user feedback, and telemetry information. Analysis of software data has a long tradition [12] in the empirical software engineering, software reliability, and mining software repositories communities. Still, according to an Accenture survey of 254 US managers in industry, up to 40 percent of major decisions are based on gut feel rather than facts [13]. With the big data boom in recent years, several research groups pushed for more use of data for decision making [4,5,6] and shared their experiences collaborating with industry on analytics projects [14,5,15]. Software analytics has been the dedicated topic of tutorials and panels at the ICSE conference [16,17], as well as special issues of IEEE Software (July 2013 and September 2013). Zhang et al. [18] emphasized the trinity of software analytics in the form of three research topics (development process, system, users) as well as three technology pillars (information visualization, analysis algorithms, large-scale computing). Buse and Zimmermann argued for a dedicated analyst role in software projects [6] and presented an empirical survey with 110 professionals on guidelines for analytics in software development [19]. They identified seven typical scenarios and ranked popular indicators among professionals. Buse and Zimmermann's focus was on the usage of analytics in general (i.e., how people make decisions), rather than on collecting and ranking individual questions, which is the focus of this paper.

Information Needs. Several researchers have identified information needs and created catalogs of frequently asked questions. (None of them focused on questions for data scientists, however.)

- Sillito et al. [20] observed 25 developers to identify a catalog of 44 questions that programmers ask during software evolution tasks.
- Phillips et al. [21] interviewed seven release managers in a large company to identify information needs related to integration decisions when releasing software.
- Ko et al. [22] observed 17 developers at Microsoft and identified 21 types of information that they need.
- Fritz and Murphy [23] interviewed 11 professional developers to identify 78 questions developers want to ask, but for which support is lacking due to lack of integration between different kinds of project information.
- Begel et al. [24] prioritized 31 different information needs about inter-team coordination in a survey among 110 software professionals.
- Breu et al. [25] identified eight categories of information needs related to the interaction between developers and users based on a sample of 600 bug reports.
- Roehm et al. [26] observed 28 developers to identify strategies they followed, information they needed, and tools they used in program comprehension tasks.

The scale of our study is larger: the findings in this paper are based on input from 800+ software professionals in many different roles across three engineering disciplines (development, testing, and program management).

3. METHODOLOGY

The findings in this paper come from two surveys:

1. An initial survey to solicit questions that software professionals would like data scientists to answer.
2. A second survey to rank and prioritize questions, as well as identify questions that are more popular among certain groups of software professionals.

This section describes the surveys and the analysis of the responses in more detail. We summarize our process in Figure 1, to which we refer with the circled numbers (❶ to ❻).

3.1 Initial Survey

Our initial survey ❶ consisted of a single question:

> *Suppose you could work with a team of data scientists and data analysts who specialize in studying how software is developed.*
>
> *Please list up to five questions you would like them to answer. Why do you want to know? What would you do with the answers?*

We additionally asked, "Why do you want to know?" and "What would you do with the answer?" in order to gain context for the response. We sent out two pilot surveys to 25 and 75 Microsoft engineers and used the responses to improve our questions. The pilot

Figure 2. Photo of some of the subcategories that emerged during the open card sort.

also demonstrated the need to seed the survey with data analytics questions. The following seed questions came from the pilot responses, Greg Wilson's list [9], and from our own experiences:

Here are examples of questions that others have already asked.

- *Do unit tests save more time in debugging than they take to write/run/keep updated?*
- *Do distributed version control systems offer any advantages over centralized version control systems?*
- *Is it really twice as hard to debug as it is to write the code in the first place?*
- *When does it make sense to reinvent the wheel vs. use an existing library?*
- *What impact does code quality have on our ability to monetize a software service?*

We sent the initial survey to 1,500 Microsoft software engineers randomly chosen by discipline in September 2012. Since monetary incentives have been found to increase the participation in surveys [27], we offered survey recipients the opportunity to enter a raffle for a $250 Visa Check Card. We received 728 items in 203 responses, for a response rate of 13.5%. This rate is comparable to the 14% to 20% [28] and 6 to 36% [28] reported for other surveys in software engineering. The respondents had a median experience of 10 years in the software industry; by discipline, 36.5% were developers, 38.9% testers, and 22.7% program managers. At Microsoft, program managers gather customer requirements, write design specifications, and manage the project schedule. Developers turn design specifications into implementation specifications, and write the code. Testers turn design specifications into testing specifications, write test harnesses, and conduct black box testing.

To group items into categories, we used an open card sort [29]. ❷ Card sorting is a technique that is widely used to create mental models and derive taxonomies from data. In our case, card sorting also helps us to deduce a higher level of abstraction and identify common themes. Cart sorting has three phases: in the *preparation* phase, we create cards for each question written by the respondents; in the *execution* phase, cards are sorted into meaningful groups with a descriptive title; finally, in the *analysis* phase, abstract hierarchies are formed in order to deduce general categories and themes. Our card sort was *open*, meaning we had no predefined groups; instead, we let the groups emerge and evolve during the sorting process (see Figure 2). By contrast, a *closed* card sort has predefined groups, which is typically used when the themes are known in advance.

Both authors jointly sorted cards over a series of several sessions until we agreed on a set of categories and subcategories. We decided to discard 49 out of the 728 cards because they only made general comments on software development and did not inquire about any topic at all. Ultimately, this resulted in 679 questions in

Table 1. The 12 categories identified in the card sort.

Category		Cards		Subcategories	Descriptive Questions
Bug Measurements	BUG	23	3%	4	7
Development Practices	DP	117	16%	13	28
Development Best Practices	BEST	65	9%	6	9
Testing practices	TP	101	14%	5	20
Evaluating Quality	EQ	47	6%	6	16
Services	SVC	42	6%	2	8
Customers and Requirements	CR	44	6%	5	9
Software Development Lifecycle	SL	32	4%	3	7
Software Development Process	PROC	47	6%	3	14
Productivity	PROD	57	8%	5	13
Teams and Collaboration	TC	73	10%	7	11
Reuse and Shared Components	RSC	31	4%	1	3
Discarded Cards		*49*	*7%*		
Total Cards		728	100%	60	145

12 categories, each with one to thirteen subcategories (Table 1 breaks this down in detail). In this paper, we focus only on the main categories (you may find the list of subcategories in a technical report [11]). We discuss the individual categories in Section 4.1. After each one, we list representative examples of respondent-provided questions that were sorted into that category.

The respondents' questions were not easy to work with; many were the same or similar to one another, most were quite verbose, and others were overly specific. We distilled them into a set of descriptive questions ❸ that more concisely described each category (and sub-category). We offer an example of respondent-provided questions and the more concise descriptive question we created.

<u>Respondent-provided questions:</u>

- *"How does the quality of software change over time – does software age? I would use this to plan the replacement of components."*
- *"How do security vulnerabilities correlate to age / complexity / code churn / etc. of a code base? Identify areas to focus on for in-depth security review or re-architecting."*
- *"What will the cost of maintaining a body of code or particular solution be? Software is rarely a fire and forget proposition but usually has a fairly predictable lifecycle. We rarely examine the long term cost of projects and the burden we place on ourselves and SE as we move forward."*

<u>Descriptive question that we distilled:</u>

- How does the age of code affect its quality, complexity, maintainability, and security?

The number of descriptive questions for each category is shown in Table 1; in total, we created 145 descriptive questions. The full list is available in our technical report [11].

3.2 The Rating Survey

In order to rank and prioritize the 145 descriptive questions, we created a second survey. ❹ However, we felt that asking respondents to rate *all* of the questions would have put too much burden on them, resulting in a low response rate and high drop-off rate [30]. Therefore, we chose a *split questionnaire survey design* [31] in which the questionnaire is split into component blocks. Each individual respondent is administered just a subset of the components. Comparisons between split questionnaires and complete surveys have found that little information is lost [31].

We defined our components to be the categories we identified in our card sort. To make them similar in size, we divided the Development Practice category in half and combined small categories together (Bug Measurements with Development Best Practices;

Productivity with Reuse and Shared Components; and Services with Software Development Lifecycle). This resulted in 10 components (blocks) of descriptive questions. Each participant in our survey was presented with two blocks chosen at random, and asked to rate each descriptive question within.

In your opinion, how important is it to have a software data analytics team answer this question?

- *Essential*
- *Worthwhile*
- *Unimportant*
- *Unwise*
- *I don't understand*

We employed an asymmetric survey response scale inspired by Prof. Noriaki Kano, who created a model for product development and customer satisfaction [32]. It measures three aspects: must have (Essential), nice to have (Worthwhile), and detraction (Unwise).

We sent the rating survey to 2,500 engineers (randomly chosen by discipline) who had not been invited to the initial survey. Program managers were oversampled (1,000 vs. 750) because of their lower response in the initial survey. Respondents were enticed to respond with the opportunity to enter into a raffle for a $250 Visa Check Card. We received 607 responses to this survey, for a response rate of 24.3%. This rate was better than our initial survey, possibly because of the simpler multiple-choice format. 29.3% of the respondents were developers, 30.1% were testers, and 40.5% were program managers. The 607 survey respondents submitted 16,765 ratings. On average, each respondent rated 27.6 questions, while each question was rated by 115.5 respondents. The respondents used most of the scale, on average, 3.5 of the five possible answers. 45% of respondents used four or all five answers, and only three respondents gave the same answer for all questions that they rated.

For the response analysis, we focused on the top- and bottom-rated questions as well as differences in ratings by demographics.

Top-Rated/Bottom-Rated Questions. ❺

We followed the advice of Kitchenham and Pfleeger [33] to dichotomize the ordinal Kano scale to avoid any scale violations. We computed the following percentages for each descriptive question:

- Percentage of "**Essential**" responses among all responses

$$\frac{\text{Essential}}{\text{Essential} + \text{Worthwhile} + \text{Unimportant} + \text{Unwise}}$$

- Percentage of "Essential" and "Worthwhile" responses among all responses (to which we refer as **Worthwhile+**)

$$\frac{\text{Essential} + \text{Worthwhile}}{\text{Essential} + \text{Worthwhile} + \text{Unimportant} + \text{Unwise}}$$

- Percentage of "**Unwise**" responses among all responses

$$\frac{\text{Unwise}}{\text{Essential} + \text{Worthwhile} + \text{Unimportant} + \text{Unwise}}$$

In addition, we computed the rank of each question based on the above percentages, with the top rank (#1) having the highest percentage in a dimension (Essential, Worthwhile+, or Unwise).

In Section 4.2 of this paper, we report the descriptive questions that are the most desired (Top 20 Essential, Top 20 Worthwhile+) and the most opposed (Top 10 Unwise). The survey results for all 145 questions are available as a technical report [11].

Rating Differences by Demographics. ❻

Our survey was not anonymous, so we could enrich the response data with the following demographics from the employee database:

- *Discipline:* Development, Testing, Program Management
- *Region:* Asia, Europe, North America, Other
- *Number of Full-Time Employees* who directly (or indirectly) report to the respondent.
- *Current Role:* Manager, if the respondent has reports (Number of Full-Time Employees > 0); Individual Contributor (IC), if the respondent has no reports.
- *Years as Manager:* The number of years that the respondent has ever worked in a management role over his tenure at Microsoft (decimals OK). Note, some ICs can also have management experience if they had been a manager at Microsoft in the past, but are not managers now.
- *Has Management Experience:* yes, if Years as Manager > 0; otherwise no.
- *Years at Microsoft:* The number of years that the respondent has been working at Microsoft (decimals OK).

To identify the demographics that influenced the responses for *individual* questions, we built stepwise logistic regression models. For each of the 145 descriptive questions, we built a model with the presence of an Essential response (yes/no) as dependent variable and the demographics above as independent variables. We built similar models for the presence of Worthwhile+ and Unwise responses, respectively. In total, we built 435 models, three for each of the 145 descriptive questions.

We used stepwise regression to eliminate demographics that did not improve the model for a given question and a given response. In addition, we removed demographics for which the coefficient in the logistic regression model was not statistically significant at $p<.01$. For example, for Question 121 "How can we make it easier for people to find and use commonly used tools?" and the response "Essential", after stepwise regression, the logistic model included the demographics Discipline=Testing, Discipline=Software Development, Number of Full Time Employees, and Has Reports=yes. However, the coefficient was only statistically significant at the 0.01 level for Discipline=Testing. In this case, we report the difference for the Discipline demographics. Our analysis revealed 29 cases where demographics responded differently to questions, which we discuss in Section 4.3.

3.3 Limitations / Threats to Validity

Program managers responded to the initial survey at a lower rate than the rating survey, generating fewer questions than they rated. To address any concerns related to sampling, we report statistically significant differences in the ratings split by discipline (Table 4).

Drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables [34]. Since our surveys were conducted with Microsoft engineers, we cannot assume that the results will generalize outside of Microsoft. However, there is nothing specific or different in the study that prevents replication at other companies or in the open source domain. Replicating our study in different organizational contexts will help generalize its results and build an empirical body of knowledge. To facilitate replication, both of our surveys and our list of 145 descriptive questions are available in a technical report [11].

There are some frequent misconceptions about empirical research within one company – it is not good enough, provides little value for the academic community, and does not contribute to scientific development. Historical evidence shows otherwise. Flyvbjerg provides several examples of individual cases that contributed to discovery in physics, economics, and social sciences [35]. Beveridge observed that for social sciences, "more discoveries have arisen from intense observation than from statistics applied to large groups” (as quoted in Kuper and Kuper [36]). Note this should not imply that research focusing on large samples or entire populations is not important. On the contrary, for the development of an empirical body of knowledge, both types of research are essential [34].

4. RESULTS

The goal of this research was to uncover the kinds of software data analytics questions that professional software engineers want to ask. While the research literature has informed our own understanding, the Microsoft software engineers we surveyed are informed by their own experiences designing, planning, building, testing, and deploying very large software products. In this section, we report on the results from our two surveys and subsequent data analyses.

4.1 Question Categories

From the 679 questions we received during our initial survey, we found out that our respondents are curious about a wide variety of topics that they would like to have a team of data scientists answer. Our card sort enabled us to group these questions into 12 categories, each described below. After each category, we list several of the respondents’ questions to illustrate the diversity of their interests.

Bug Measurements (BUG) – Respondents were curious about many aspects of software bugs: where they are found in code and in the product design, which bugs are most commonly made by developers, where in the application lifecycle they are caught, both before and after the product ships, how much it costs to fix bugs, and how well tool automation helps to mitigate them. In addition, some respondents were curious about whether bug counts should be used to measure product quality and developer effectiveness.

- *“For each bug, at what stage in the development cycle was the bug found, at what stage was it introduced, at what stage could it have been found?”*
- *“Are there categories of mistakes that people make that result in hotfixes after the fact? For example, date/time mistakes, or memory allocation mistakes, etc.”*
- *“Have you ever had your spec document worked through thoroughly by QA or a computer scientist/theorist to detect every single edge case beforehand? Did it help? Was it worth the effort?”*

Development Practices (DP) – Respondents asked about all kinds of code-oriented practices, including debugging, performance profiling, refactoring, branching in repositories, code reviewing, commenting, and documenting code. In addition, respondents expressed concern about development costs due to legacy code, execution telemetry, customer telemetry, inaccurate effort estimation, and product risk caused by code changes.

- *“How can I judge when adding telemetry/instrumentation to code will pay off vs. when it will just tell me things I already know?”*
- *“We are often told that code comments are important to assist others understand code, but is this really true?”*
- *“Do many eyes make bugs shallow? Where is the break-even?”*
- *“What is the net cost of refactoring legacy code so that it can be covered with unit tests, compared to leaving legacy code uncovered and making changes in it?”*
- *“What are the provably best metrics to track during design/planning to determine project complexity and cost? What is the expected margin of error?”*

Development Best Practices (BEST) – This category resembles the *Development Practices* category above, but respondents were specifically interested in the best (or worst) way to conduct a software practice. For example, respondents want to know the right technique for migrating between software versions, the best way to track work items, the right time to use formal methods for software analysis, or which criteria should influence the decision to use a particular programming language or API.

- *“What are the best and worst practices [of] teams that miss deadlines, or slip beyond their release schedule call out? It would be great to note what practices teams adopt and/or cut when they are feeling pressured to meet certain deadlines.”*
- *“Could machine learning be used to provide better coding guidelines and best practices?”*
- *“How many times does a piece of code [get] re-written? Is it better to write code quickly and then iterate on it, or do a longer process of planning and just write it ‘once’?”*
- *“Have you ever written a loop invariant to prove that your algorithm terminates? Did it help?”*
- *“What's the recommended way to detect/locate PII (personally identifiable information) inside arbitrary data? Are there any guidelines on how to detect it (e.g. shared rulesets)?”*
- *“What are the time benefits to strict coding standards when code maintenance is inherited by another dev?”*
- *“Which coding guidelines/patterns have the most effect on code quality (e.g. small functions, lower complexity metrics, removal of duplicate code)?”*
- *“How long does it take to gain a payoff from moving to new software-writing tools?”*

Testing Practices (TP) – Testing was the most diverse category, covering test automation, testing strategies, unit testing, test-driven development, test coverage, and test case elimination. In addition, respondents were interested in test processes, for example, writing tests: who should do it and when should it happen, sharing tests and test infrastructures across teams, and measuring the effectiveness of particular kinds of tests in finding impactful bugs.

- *“What is the cost/benefit analysis of the different levels of testing i.e. unit testing vs. component testing vs. integration testing vs. business process testing?”*
- *“How much percentage of development time needs to be spent on unit testing to ensure quality? At what point does it become too much time spent on writing test for every line?”*
- *“What's the useful lifetime of a test? At what point does the test become so unlikely to uncover an issue that it isn't worth the maintenance and infrastructure cost of keeping it running?”*
- *“How many spec/code issues were found thanks to the creation of a Test Design Spec?”*
- *“What value is there from investigating regression data to improve future code quality if the regressions could not be found internally?”*

Evaluating Quality (EQ) – Respondents’ questions included code optimization tradeoffs, the best metrics for deciding whether to ship software, complexity metrics, code duplication, the relationship between test coverage and customer feature usage, legacy and aging codebases, and the most common causes of poor software quality. This category is perhaps the most similar to what one would find in the research literature.

- *“Some means to get a relation defined between crashes found in real world and tests in our collateral which hit the same code path. Knowing we hit this code path but still we have a crashing bug in there would be good to know.”*
- *“Is there a difference in code coverage and number of actionable code bugs with respect to tests developed in native Win32 vs. managed code?”*
- *“Do teams who leverage code coverage for more than reporting coverage percent to management deliver higher-quality software?”*
- *“How much self-hosting/beta time does a feature of a certain complexity need before it can be considered reliable?”*
- *“How much do existing bugs in a code base affect the velocity with which new features can be added to that code base?”*

Services (SVC) – Many questions concerned developing software for the cloud. Some concerned how to change development and testing practices when migrating software to the cloud, while others referenced operations and performance measurement. A set of respondents were interested in the effects on customer retention and monetization caused by moving to more frequent, smaller releases from larger, less frequent releases.

"What are the KPIs that are used in managing our on-line services today? This should be as standardized as possible."

"Do distributed systems scale better than centralized systems with vertical/horizontal scaling?"

"When does it make sense to change a helper library to a service of its own?"

"What's the best approach for testing in production, without affecting the live production data and compromising the performance, while being able to test and regress all the functionality?"

"For online service, agility and code quality are always what we want to achieve. Which is more important?"

Customers and Requirements (CR) – This category contained the majority of the concerns about customer interests, ultimately focusing on how customers might react to various development tradeoffs, for example, the most liked and important features, the necessity of backward compatibility, or the impact of testing in production. Respondents were also interested in the payoff that arises from investing in specifications to various degrees, and of the costs associated with increasing customer input into software design.

"How many features of the s/w are used by people and at what percentage? It will help me understand what is the impact/ROI on adding a new complex feature in terms of the investment we put to add new feature vs. its impact on user."

"How do new free online versions of apps impact sales and revenue of the full pay versions?"

"Does Testing in Production hurt the reputation or perception of quality of product?"

"What factors should be considered before deprecating features?"

"How do we determine what problems our customers are having with our software? I would like to know what is failing in our software when our customers use it."

"Is there a correlation between the level of detail in a functional/requirements spec and the accuracy of dev estimates based on it?"

Software Development Lifecycle (SL) – All of the questions in this category regarded time as a vital factor in designing the software lifecycle. For example, respondents wanted to find out how development time should be allocated between planning, design, coding, and testing, and the impact of this allocation could have on the software.

"From an efficiency standpoint, how detailed should dev design docs be before just starting to code?"

"How do we monetize the intended savings when a decision is taken to make a change in product design? For example, if we mitigate a launch risk, how do we figure out the money saved if the risk never become an issue? i.e. If I have $150 to spend, should I spend it on back up sensors or bigger mirrors? Which investment would save me more money? How do I determine that?"

"What has a bigger impact on overall product quality, up front standards and best practices for dev, or longer testing and bake time before product release?"

"Do shorter release cycles result in a better quality product? A more feature-filled product?"

Software Development Process (PROC) – The choice of software methodology was on respondent's minds when they answered this question. In what ways is Agile better than Waterfall? What are the benefits of pair programming? Do particular methodologies work better for cloud services than for shrink-wrapped products? Others were interested in whether Microsoft would benefit from a company-wide software process and tool chain, and how to manage it.

"How is Agile process model more or less efficient in when it comes to number of bugs, meeting deadlines, customer satisfaction, work-life balance of Dev/Test/PM?"

"Which software development model(s) are most successful in producing great software?"

"Is it more efficient for the company to have one consistent process for building software that all projects use, rather than each team/division/project doing it their own way?"

"In an organization that is running Scrum (or any agile flavor) development cycles, how does it meet the challenge of leaving time for innovation?"

Productivity (PROD) – For many respondents, this category is what they think of when they hear the term "software data analytics," and what many of them feel is the fear of the consequences caused by being unfairly judged by management. Questions in this category included investigating the quality of many different proxy metrics for measuring the productivity of software developers, and understanding the relationship between individual productivity and team or product success. Some respondents wanted to know how to monitor their own or their team's productivity, for instance, when learning a new codebase, or working with a new team member or dealing with the issues caused for a team member leaving.

"Are there measurable differences in productivity between experienced developers with CS degrees and developers with unrelated degrees or no degrees?"

"Are daily Scrum status meetings effective?"

"Does peer [sic] programming improve code quality and throughput?"

"How does productivity vary between open work spaces and traditional offices?"

"Does churn in organizations cause software development to slow? As people move between orgs, do we see a slowdown in productivity?"

"Are more senior staff actually more productive?"

"How do we measure the productivity of our engineers? Which engineer is more productive?"

"How do we measure the productivity or business savings with a specific feature or toolset?"

"Does a person's language, age, experience or gender affect the bugs s/he may introduce/find?"

"How well am I utilizing my resources? What trace is left by activities per person? Not just code changes, but bug database changes, doc repository changes, etc."

Teams and Collaboration (TC) – Respondents here cared about team makeup, i.e. how many people of various development roles should be on a team, and how big it should be for a given feature size. A second aspect of the questions wanted to learn which practices could improve sharing and collaboration within and between teams, especially relating to shared knowledge, code ownership, task status, and commonly used tools.

"How can we share knowledge more effectively to code faster? There're a lot of people who already have the knowledge, but don't have the time to share it or just don't care to share it."

"Is integrated engineering (i.e. elimination of the Test organizations) a better use of budgets and resources than having Dev, Program Management, and Test?"

"How can I find the best developer contact for a piece of apparently 'abandoned' code?"

"What is the optimal team size N for a feature of scope M?"

"How much time/week do you spend updating/sending/reporting status to other people?"

"How does one improve relations when doing cross-org collaborations?"

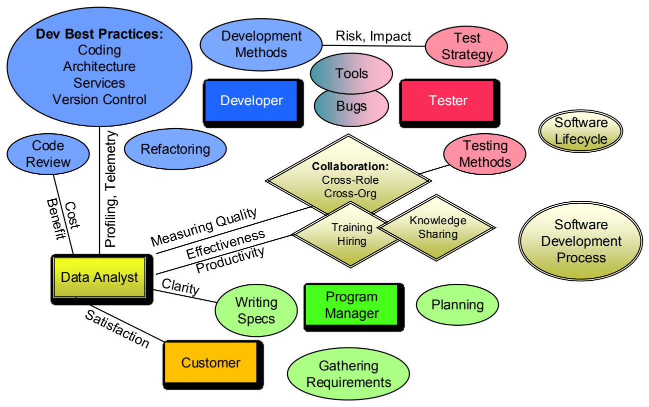


Figure 3. Relationship between card sort categories.

Reuse and Shared Components (RSC) – This was the smallest category to come from our card sort, and concerned one main issue: when should code be written from scratch vs. reused from another codebase? Respondents wanted to know the cost of searching for the right code, and the potential impact caused by just linking to it vs. copying it wholesale.

"Reusing an existing library is always good and easy. What's the best way to reuse an existing library if it matches the functionality we are looking for, but does not match the architecture of our project?"
"When is it best to create your own UI model rather than adopt a standard that might be familiar, but not fit your use exactly right?"
"Do shared codebases actually reduce developer productivity and innovation rates at the company?"
"How can I find out if someone else has already solved a similar problem before, e.g., caching, datatype conversion etc.?"

Category Relationships – The categories exist in a set of relationships, having to do with people and the activities that connect them. In Figure 3, we show an affinity chart with five job roles (drawn as boxes) that were mentioned in the respondents' questions: *developer*, *tester*, *program manager*, *customer*, and *data analyst*. Nearby each role are ovals listing popular activities done by people in those roles, the attributes of which drew the attention of the respondents. The lines drawn from the data analyst link her investigation and analyses to various development activities (some of the activities are linked together themselves). The three diamonds in the center of the figure indicate that *collaboration, knowledge sharing,* and *training* are three aspects that affect personnel in all job roles. Finally, shapes drawn with a double border indicate crosscutting attributes—they apply across all of the roles, the activities, and the entire software product and development process itself.

4.2 Top-Rated/Bottom-Rated Questions

Table 2 shows the questions with the highest percentages for the "Essential" and "Worthwhile or higher" (Worthwhile+) responses. In the table, we show the Top 20 questions for both *Essential* and *Worthwhile+*, however, they overlap significantly. As a result, we only show 24 questions. In addition to the question, the table shows the percentage of respondents who selected Essential, Worthwhile+, and Unwise, as well as their corresponding ranks. The table is sorted in decreasing order by the Essential percentages.

One thing we noticed about the Top 20 list is the striking fraction of questions that are concerned about customers. Nine questions (marked with icon 👪, Q27, Q18, Q28, Q66, Q19, Q131, Q92, Q25, and Q17), including the top two, demonstrate a concern by program managers, developers *and* testers that they plan, design, implement, test, and profile the features that match what the customers expect and use, and benefit the customers the most. Another eight questions (marked with icon 🛠, Q115, Q74, Q84, Q100, Q83, Q2, Q3, and Q102) focus on the engineer and the effects of software development practices and processes on her work. The remaining seven questions (marked with icon ★, Q50, Q86, Q40, Q42, Q59, Q1, and Q60) are chiefly concerned with product quality issues.

In Table 3, we show the 10 questions with the most opposition, by which we mean the highest percentages for the "Unwise" response (sorted in decreasing order by the *Unwise* percentage).

Of the questions with the most opposition in Table 3, the top five are about the fear that respondents had of being ranked and rated. This may be due to the impact these ratings could have on the employee's yearly performance review, which include the possibility of job termination for poor performance. This fear has been seen before in companies that employ employee metrics collection and evaluation systems [37].

The other questions deemed unwise to be answered explore situations deemed improbable (and unwise), for example achieving 100% test coverage is not often possible or helpful even when it happens. Second, Microsoft's feature teams employ a freedom of software tools and process that enables them to develop their own practices to suit their needs without requiring adherence to top-down engineering management mandates. This explains why Q112 and Q113 are on the list as well. Finally, the last two questions could be perceived as tautological statements by respondents—they might think that there is only one correct answer to each (e.g. Q34: Always, and Q24: As much as you can afford), which may or may not be true.

Looking at the percentages of *Essential* in the Top 20 rankings, we notice that there is a steep drop from the Top 3 questions to the remainder. However, considering *Worthwhile* responses as well, we see a much smoother decline, in fact, for 140 questions, more than 50% of responses were Worthwhile or higher (Worthwhile+), indicating that many of the questions on the survey were worth having a team of data scientists look into the answers.

4.3 Rating Differences by Demographics

Table 4 shows demographics for which we identified statistically significant differences in the ratings of the questions. The table is grouped by Discipline (Developer, Tester, and Program Manager), Management Role (Manager and Individual Contributor), Work Region (North America, Europe, and Asia), Years of Experience as a Manager, and Years Working at the Company.

In addition to the question, we list the survey response for which the difference was observed (in column *Response*) and show (in the final column) how the question was rated differently by the respective demographic. The demographic with the highest rating (for the first three demographics) is highlighted in bold.

To explain further, please look at the first entry of the table. Question 5, "How many new bugs are introduced for every bug that is fixed?" was rated by each *Discipline* differently with respect to the survey response *Essential.* Respondents in the Test discipline were more likely to rate the question as Essential (41.9%) than respondents in the Development (27.3%) or Program Management (12.5%) disciplines.

For demographics related to years of experience (Years as Manager, Years at Company), we report the change in odds per year. Look at the first entry under Years as Manager. For every year that someone spent as a manager, the odds that they rated Question 85,

Table 2. Questions with the highest "Essential" and "Worthwhile or higher" percentages.

				Percentages			Rank		
		Question	Category	**Essential**	Worthwhile+	Unwise	**Essential**	Worthwhile+	Unwise
👥	Q27	How do users typically use my application?	DP	80.0%	99.2%	0.83%	1	1	128
👥	Q18	What parts of a software product are most used and/or loved by customers?	CR	72.0%	98.5%	0.00%	2	2	130
★	Q50	How effective are the quality gates we run at checkin?	DP	62.4%	96.6%	0.85%	3	6	125
⚒	Q115	How can we improve collaboration and sharing between teams?	TC	54.5%	96.4%	0.00%	4	8	130
★	Q86	What are best key performance indicators (KPIs) for monitoring services?	SVC	53.2%	93.6%	0.92%	5	12	106
★	Q40	What is the impact of a code change or requirements change to the project and tests?	DP	52.1%	94.0%	0.00%	6	10	130
⚒	Q74	What is the impact of tools on productivity?	PROD	50.5%	97.2%	0.92%	7	4	106
⚒	Q84	How do I avoid reinventing the wheel by sharing and/or searching for code?	RSC	50.0%	90.9%	0.91%	8	19	108
👥	Q28	What are the common patterns of execution in my application?	DP	48.7%	96.6%	0.84%	9	5	127
👥	Q66	How well does test coverage correspond to actual code usage by our customers?	EQ	48.7%	92.0%	0.00%	10	16	130
★	Q42	What tools can help us measure and estimate the risk associated with code changes?	DP	47.8%	92.2%	0.00%	11	15	130
★	Q59	What are effective metrics for ship quality?	EQ	47.8%	96.5%	1.77%	12	7	91
⚒	Q100	How much do design changes cost us and how can we reduce their risk?	SL	46.6%	94.8%	0.86%	13	9	123
👥	Q19	What are the best ways to change a product's features without losing customers?	CR	46.2%	92.3%	1.54%	14	14	103
👥	Q131	Which test strategies find the most impactful bugs (e.g., assertions, in-circuit testing, A/B testing)?	TP	44.5%	91.8%	0.91%	15	18	108
⚒	Q83	When should I write code from scratch vs. reuse legacy code?	RSC	44.5%	84.5%	3.64%	15	45	37
	Q1	What is the impact and/or cost of findings bugs at a certain stage in the development cycle?	BUG	43.1%	87.9%	2.59%	17	28	68
👥	Q92	What is the tradeoff between releasing more features or releasing more often?	SVC	42.5%	79.6%	0.00%	18	64	130
⚒	Q2	What kinds of mistakes do developers make in their software? Which ones are the most common?	BUG	41.7%	98.3%	0.00%	19	3	130
👥	Q25	How important is a particular requirement?	CR	41.7%	87.4%	2.36%	20	32	73
★	Q60	How should we use metrics to help us decide when a feature is good enough to release (or poor enough to cancel)?	EQ	41.1%	90.2%	3.57%	23	20	41
👥	Q17	What is the best way to collect customer feedback?	CR	39.8%	93.0%	1.56%	26	13	101
⚒	Q3	In what places in their software code do developers make the most mistakes?	BUG	35.0%	94.0%	0.00%	41	10	130
★	Q102	What kinds of problems happen because there is too much software process?	PROC	30.1%	92.0%	0.88%	57	16	116

Table 3. Questions with the highest "Unwise" percentage (opposition).

			Percentages			Rank		
	Question	Category	Essential	Worthwhile+	**Unwise**	Essential	Worthwhile+	**Unwise**
Q72	Which individual measures correlate with employee productivity (e.g., employee age, tenure, engineering skills, education, promotion velocity, IQ)?	PROD	7.3%	44.5%	25.5%	142	141	1
Q71	Which coding measures correlate with employee productivity (e.g., lines of code, time it takes to build the software, a particular tool set, pair programming, number of hours of coding per day, language)?	PROD	15.6%	56.9%	22.0%	126	131	2
Q75	What metrics can be used to compare employees?	PROD	19.4%	67.6%	21.3%	110	112	3
Q70	How can we measure the productivity of a Microsoft employee?	PROD	19.1%	70.9%	20.9%	113	104	4
Q6	Is the number of bugs a good measure of developer effectiveness?	BUG	16.4%	54.3%	17.2%	122	136	5
Q128	Can I generate 100% test coverage?	TP	15.3%	44.1%	14.4%	127	142	6
Q113	Who should be in charge of creating and maintaining a consistent company-wide software process and tool chain?	PROC	21.9%	55.3%	12.3%	93	134	7
Q112	What are the benefits of a consistent, company-wide software process and tool chain?	PROC	25.2%	78.3%	10.4%	79	72	8
Q34	When are code comments worth the effort to write them?	DP	7.9%	41.2%	9.6%	141	143	9
Q24	How much time and money does it cost to add customer input into your design?	CR	15.9%	68.2%	8.3%	124	111	10

"How much cloned code is ok to have in a codebase?" with the response *Essential*, increased by 36% in the survey.

Discipline. Many of the differences here agree with common sense. Testers are more interested in test suites, bugs, and product quality, while developers care about migrating code between versions of a library (as also noted by McCamant and Ernst [38]). Testers also cared about helping people find commonly used tools, the cost of customer input, and identifying someone to unify the company's tools and processes.

Management Role. More individual contributors than managers are interested in finding the legacy code in their systems and discovering the optimal time in the development cycle to test their application's performance. It was surprising that more individual contributors than managers want to find out how to measure an employee's productivity (the fourth-ranked *Unwise* question). We would have expected to see the opposite result. Managers are interested in finding the most commonly used tools on software teams, possible to reduce cost and to improve their team's productivity.

Region. Half of the respondents working in Asia (mainly India and China) reported they wanted data scientists to look into employee productivity. One possible explanation may be that employees in Asia believe that striving for top personal performance is the best

Table 4. Statistically significant rating differences by demographics.
The demographic with the highest rating is highlighted in bold. Questions that are also in Table 2 are shown in *italics.*

	Question	Category	Response	Discipline		
				Dev	Test	PM
Q5	How many new bugs are introduced for every bug that is fixed?	BUG	Essential	27.3%	**41.9%**	12.5%
Q10	When should we migrate our code from one version of a library to the next?	BEST	Essential	**32.6%**	16.7%	5.1%
Q20	How much value do customers place on backward compatibility?	CR	Essential	14.3%	**47.1%**	18.3%
Q21	What is the tradeoff between frequency and high-quality when releasing software?	CR	Essential	22.9%	**48.5%**	14.5%
Q42	*What tools can help us measure and estimate the risk associated with code changes? (Essential #11)*	DP	Essential	34.5%	**70.6%**	40.4%
Q121	How can we make it easier for people to find and use commonly used tools?	TC	Essential	27.3%	**48.6%**	20.0%
Q24	*How much time and money does it cost to add customer input into your design? (Unwise #10)*	CR	Worthwhile+	62.9%	**88.2%**	60.3%
Q113	*Who should be in charge of creating and maintaining a consistent company-wide software process and tool chain? (Unwise #7)*	PROC	Worthwhile+	60.0%	**71.9%**	40.4%
Q132	How should we handle test redundancy and/or duplicate tests?	TP	Worthwhile+	48.6%	**81.1%**	47.4%
Q134	Should we develop a separate test suite for servicing a product after we ship it?	TP	Worthwhile+	32.4%	**67.6%**	**67.6%**

				Management Role	
				Manager	Ind. Contributor
Q29	How much legacy code is in my codebase?	DP	Worthwhile+	36.7%	**65.2%**
Q31	When in the development cycle should we test performance?	DP	Worthwhile+	63.3%	**81.4%**
Q70	*How can we measure the productivity of a Microsoft employee? (Unwise #4)*	PROD	Worthwhile+	57.1%	**77.3%**
Q120	What are the most commonly used tools on a software team?	TC	Worthwhile+	**95.8%**	67.8%

				Region		
				Asia	Europe	North America
Q70	*How can we measure the productivity of a Microsoft employee? (Unwise #4)*	PROD	Essential	**52.9%**	30.0%	11.0%
Q104	How do software methodologies affect the success and customer satisfaction of shrinkwrapped and service-oriented products?	PROC	Essential	**52.9%**	10.0%	24.7%
Q128	*Can I generate 100% test coverage? (Unwise #6)*	TP	Essential	**60.0%**	0.0%	9.0%
Q129	What is the effectiveness, reliability, and cost of automated testing?	TP	Essential	**71.4%**	12.5%	23.6%

				Years as Manager
				change in odds per year
Q85	How much cloned code is ok to have in a codebase?	RSC	Essential	36%
Q69	How does the age of code affect its quality, complexity, maintainbility, and security?	EQ	Worthwhile+	-28%

				Years at Company
				change in odds per year
Q16	What criteria should we decide when to use managed code or native code (e.g., speed, productivity, functionality, newer language features, code quality)?	BEST	Essential	-23%
Q119	What are the best tools and processes for sharing knowledge and task status?	TC	Essential	-18%
Q142	Should we do Test-Driven Development?	TP	Essential	-19%
Q95	How much time went into testing vs. into development?	SL	Worthwhile+	-12%
Q123	How much distinction should there be between developer/tester roles?	TC	Worthwhile+	-14%
Q136	Who should write unit tests, developers or testers?	TP	Worthwhile+	-13%
Q142	Should we do Test-Driven Development?	TP	Worthwhile+	-13%
Q144	How should we do Test-Driven Development while prototyping?	TP	Worthwhile+	-14%
Q113	*Who should be in charge of creating and maintaining a consistent company-wide software process and tool chain? (Unwise #7)*	PROC	**Unwise**	15%

way to get ahead in one's career and uncovering the secret to improved productivity is one way to improve their performance. Testing questions also interested the employees from Asia likely due to the large fraction of software testing work they do there.

Years as Manager. In the survey responses, managers with more experience cared less about investigating the impact of old code on product quality than those with less experience. One confounding factor here is that teams that work on older software products (e.g., Windows, Office) usually have more experienced managers leading them, while less experienced managers may be more involved in newer products (e.g., various web services).

Years at Company. Finally, the more experience an employee has at Microsoft, the less they are interested in having a data science team explore the managed/native code debate (often a religious-type argument with no consensual answer possible), test-driven development (another religious battle), or the preferred functions of people in testing and development roles (again, a religious battle). Employees who have been at Microsoft longer have better knowledge-sharing networks, and thus would not need to rely on specialized tools as much to help them. Finally, the more experience a person has at the company, the less likely they are to want someone to look into who should create and maintain a standardized process and tool chain – a tacit acceptance of the status quo.

5. IMPLICATIONS

The results from our surveys can serve to guide future work agendas for academic researchers, industry practitioners, and data science educators.

5.1 Research

One of the challenges faced in academic research is to choose a research topic that is novel, feasible, and impactful. While selection for novelty and feasibility are well understood and controllable, impact often requires a partnership with external players from the software industry. When the partnership is successful, these players can serve as test beds for research validation, experimentation, new tools, and ultimately, amplification of the researcher's ability to spread his or her ideas and tools widely to software practitioners. Numerous studies have reviewed the performance of such partnerships to describe the impact of research and glean the attributes that make for successful technology transfer [39,40]. One of the main findings is that to be successful, the researchers need to discover and work on the problems that industry needs to be solved.

In this paper, we have done the first part of that critical task, by identifying the questions that Microsoft, a very large, globally distributed company working on devices and services would like to see answered by data scientists knowledgeable about how software is developed. The questions in this paper can help guide academic researchers towards novel research problems to be studied and analyzed, as well as algorithms, processes, and tools to be built and tested. By building new tools, researchers can easily disseminate their ideas and make it possible for others to understand and discuss the data analyses that support them.

This should provide researchers a head start on the path to arrange academic-industry collaborative projects that address industry's software analysis needs. As other software companies replicate our study, the potential to increase the diversity of new avenues of research will grow and provide increasing opportunities for researchers to enjoy impactful research.

5.2 Practice

Software practitioners often face the challenge of building the right set of tools to help their business grow, without incurring too much cost to gather the required data, analyze it, and build tools to act on the results. Our results point at the kinds of data that should be regularly and sustainably collected in order to conduct data analyses that answer the engineers' questions. We do wish to stress that just because the survey respondents asked a question does not mean that Microsoft (or academia) does not have tools and techniques to find the answers. However, any preexisting tools that could solve these problems may have been unknown to or unusable by some of those respondents. Thus, even the presence of a question indicates an opportunity to build a better, simpler, more applicable tool, and/or find another way to distribute it to engineers.

Industrial practitioners from large, long-lived institutions have an added challenge of answering all of these questions at scale. For example, a recent study at Microsoft looked at failure and usage data from over a million computers to discover the difference in usage characteristics between pre-release and post-release failures [41]. An older study explored the linguistic attributes of identifiers in source code by analyzing over 45 million lines of code for Windows 2003 Server [42]. Mockus analyzed 18 years of change management system data and 7 years of organizational reporting data at Avaya to discover the effects of organizational volatility on software reliability [43]. Academic researchers can help with these tasks by applying their work to large corpora [44], but it is rare to find objects of study outside of industry that even come close to the size of typical industrial projects. Industry practitioners working together with academics could enable each to scale their data analyses and tools to work at the scale that industry requires.

5.3 Education

The set of 145 descriptive questions that we include in this paper can be useful to undergraduate and graduate educators who teach computer science, statistics, and data science courses. Computer science students commonly hold wildly inaccurate preconceptions about the character of the work they will encounter in their future software engineering careers [45,46]. Even beginning professional software engineers struggle to adapt the practices and processes they learned in university to the realities they find in industry [47,48]. This has not gone unnoticed by the industry, either [49,50].

The categories and diversity of the questions collected in this paper can be used in software engineering courses to illustrate real-life challenges that professional software developers face when building large, long-lived products for a variety of customers, including consumers, enterprises, and government institutions. In addition, the particular scenarios envisioned in the questions can be used to contextualize the pedagogical materials created and shared by educators to motivate and excite students with challenges faced by professional software engineers, rather than the typical problems faced by Alyssa P. Hacker and Ben Bitdiddle [51].

Our corpus of questions can inform data science and statistics educators of particular analyses and techniques that are vitally important to the workings of, for example, decision support systems that help managers ensure a successful product release, or of employee productivity measurement tools that can accurately and fairly evaluate (and motivate) an engineering workforce. One can see from the examples of the respondents' questions a range of sophistication in their phrasing and clarity. If these were students, one would hope for a coordinated set of lesson plans that would result in an increased ability to pose high-quality, specific, and actionable questions, as well as write coherent reports and follow up with the teams that needed the answers to help them continuously implement and measure process and/or tool improvements.

6. CONCLUSION

To understand the questions that software engineers would like to ask data scientists about software, we conducted two surveys: the first survey solicited questions and the second survey ranked a set of questions. The result of our analysis of the survey responses is a catalog of 145 questions grouped into 12 categories, as well as a ranking of the importance of each question. This catalog can help researchers, practitioners, and educators to improve their focus and efforts on topics that are important to large software companies.

We hope that this paper will inspire similar research projects. In order to facilitate replication of this work for additional engineering disciplines (e.g., user experience, operations, build/release, technical support) and companies, we provide the full text of both surveys as well as the 145 questions in a technical report [11].

With the growing demand for data scientists, more research is needed to better understand how people make decisions in software projects and what data and tools they need. There is also a need to increase the data literacy of future software engineers. Lastly, we need to think more about the consumer of analyses and not just the producers of them (data scientists, empirical researchers).

7. ACKNOWLEDGMENTS

Thanks to everyone who responded to one of our surveys, as well as Christian Bird, Robin Moeur, Danyel Fisher, Emerson Murphy-Hill, Jim Witschey, and Nachiappan Nagappan.

8. REFERENCES

[1] May, T. *The New Know: Innovation Powered by Analytics*. Wiley, 2009.

[2] Davenport, T.H., Harris, J.G., and Morison, R. *Analytics at Work: Smarter Decisions, Better Results*. Harvard Business Review Press, 2010.

[3] Lewis, M. *Moneyball: The Art of Winning an Unfair Game*. W. W. Norton & Company, 2003.

[4] Hassan, A.E. and Xie, T. Software intelligence: the future of mining software engineering data. In *FOSER '10: Proceedings of the Workshop on Future of Software Engineering Research* (2010), 161-166.

[5] Zhang, D., Dang, Y., Lou, J.-G., Han, S., Zhang, H., and Xie, T. Software Analytics as a Learning Case in Practice: Approaches and Experiences. In *MALETS '11: Proceedings International Workshop on Machine Learning Technologies in Software Engineering* (2011).

[6] Buse, R.P.L. and Zimmermann, T. Analytics for software development. In *FOSER '10: Proceedings of the Workshop on Future of Software Engineering Research* (2010), 77-80.

[7] Davenport, T.H. and Patil, D.J. Data Scientist: The Sexiest Job of the 21st Century. *Harvard Business Review* (October 2012).

[8] MCKINSEY GLOBAL INSTITUTE. *Big data: The next frontier for innovation, competition, and productivity*. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation. 2011.

[9] Wilson, G. *Ten Questions for Researchers*. http://neverworkintheory.org/2012/08/22/ten-questions-for-empirical-researchers.html. 2012.

[10] Meyer, B. *Empirical answers to fundamental software engineering questions*. http://bertrandmeyer.com/2013/08/29/empirical-answers-to-fundamental-software-engineering-questions/. 2013.

[11] Begel, A. and Zimmermann, T. *Appendix to Analyze This! 145 Questions for Data Scientists in Software Engineering*. Technical Report, MSR-TR-2013-84, http://aka.ms/145Questions, Microsoft Research, Redmond, WA, USA, 2013.

[12] Menzies, T. and Zimmermann, T. Software Analytics: So What? *IEEE Software*, 30, 4 (July 2013), 31-37.

[13] Accenture. *Most U.S. Companies Say Business Analytics Still Future Goal, Not Present Reality*. http://newsroom.accenture.com/article_display.cfm?article_id=4777. 2008.

[14] Lou, J.-G., Lin, Q.W., Ding, R., Fu, Q., Zhang, D., and Xie, T. Software Analytics for Incident Management of Online Services: An Experience Report. In *ASE '13: Proceedings of the Internation Conference on Automated Software Engineering* (2013).

[15] Menzies, T., Bird, C., Zimmermann, T., Schulte, W., and Kocaganeli, E. The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining. In *MALETS '11: Proceedings International Workshop on Machine Learning Technologies in Software Engineering* (2011).

[16] Zhang, D. and Xie, T. Software analytics: achievements and challenges. In *ICSE '13: Proceedings of the 2013 International Conference on Software Engineering* (2013), 1487-1487.

[17] Zhang, D. and Xie, T. Software Analytics in Practice. In *ICSE '12: Proceedings of the International Conference on Software Engineering.* (2012), 997.

[18] Zhang, D., Han, S., Dang, Y., Lou, J.-G., Zhang, H., and Xie, T. Software Analytics in Practice. *IEEE Software*, 30, 5 (September 2013), 30-37.

[19] Buse, R.P.L. and Zimmermann, T. Information needs for software development analytics. In *ICSE '12: Proceedings of 34th International Conference on Software Engineering* (2012), 987-996.

[20] Sillito, J., Murphy, G.C., and Volder, K.D. Asking and Answering Questions during a Programming Change Task. *IEEE Trans. Software Eng.*, 34, 4 (2008), 434-451.

[21] Phillips, S., Ruhe, G., and Sillito, J. Information needs for integration decisions in the release process of large-scale parallel development. In *CSCW '12: Proceedings of Computer Supported Cooperative Work* (2012), 1371-1380.

[22] Ko, A.J., DeLine, R., and Venolia, G. Information Needs in Collocated Software Development Teams. In *ICSE '07: Proceedings of International Conference on Software Engineering* (2007), 344-353.

[23] Fritz, T. and Murphy, G.C. Using information fragments to answer the questions developers ask. In *ICSE '10: Proceedings of the International Conference on Software Engineering* (2010), 175-184.

[24] Begel, A., Khoo, Y.P., and Zimmermann, T. Codebook: Discovering and exploiting relationships in software repositories. In *ICSE '10: Proceedings of International Conference on Software Engineering* (2010), 125-134.

[25] Breu, S., Premraj, R., Sillito, J., and Zimmermann, T. Information needs in bug reports: improving cooperation between developers and users. In *CSCW '10: Proceedings of Computer Supported Cooperative Work* (2010), 301-310.

[26] Roehm, T., Tiarks, R., Koschke, R., and Maalej, W. How do professional developers comprehend software? In *ICSE'12: Pro. of Intl. Conf. on Software Engineering* (2012), 255-265.

[27] VanGeest, J.B., Johnson, T.P., and Welch, V.L. Methodologies for improving response rates in surveys of physicians: a systematic review. *Evaluation and the Health Professions*, 30, 4 (December 2007), 303-321.

[28] Smith, E., Loftin, R., Murphy-Hill, E., Bird, C., and Zimmermann, T. Improving Developer Participation Rates in Surveys. In *CHASE '13: Proceedings of Workshop on Cooperative and Human Aspects of Software Engineering* (2013).

[29] Hudson, W. Card Sorting. In Soegaard, M. and Dam, R.F., eds., *Encyclopedia of Human-Computer Interaction, 2nd Ed.* The Interaction Design Foundation, 2013.

[30] Chudoba, B. *Does Adding One More Question Impact Survey Completion Rate?* http://blog.surveymonkey.com/blog/2010/12/08/survey_questions_and_completion_rates/, SurveyMonkey.com, 2010.

[31] Design, A.S.Q.S. Trivellore E. Raghunathan and James E. Grizzle. *Journal of the American Statistical Association*, 90, 429 (March 1995), 54-63.

[32] Kano, N., Seraku, N., Takahashi, F., and Tsuji, S. Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control (in Japanese)*, 14, 2 (April 1984), 39–48.

[33] Kitchenham, B. and Pfleeger, S. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*. Springer, 2007.

[34] Basili, V.R., Shull, F., and Lanubile, F. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25, 4 (July/August 1995), 456-473.

[35] Flyvbjerg, B. Five misunderstandings about case-study research. *Qualitative inquiry*, 12, 2 (2006), 219-245.

[36] Kuper, A. and Kuper, J. *The Social Science Encyclopedia*. Routledge, 1985.

[37] Umarji, M. and Seaman, C. Gauging acceptance of software metrics: Comparing perspectives of managers and developers. In *ESEM '09: Proceedings of the International Symposium on Empirical Software Engineering and Measurement* (2009), 236--247.

[38] McCamant, S. and Ernst, M.D. Predicting problems caused by component upgrades. In *ESEC/FSE '03: Proceedings of the European Software Engineering Conference / ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2003), 287--296.

[39] Gorschek, T., Wohlin, C., Carre, P., and Larsson, S. A Model for Technology Transfer in Practice. *IEEE Software*, 23, 6 (2006), 88-95.

[40] Jeffery, R. and Scott, L. Has twenty-five years of empirical software engineering made a difference? In *APSEC '02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference* (2002), 539-546.

[41] Li, P.L., Kivett, R., Zhan, Z., Jeon, S.-e., Nagappan, N., Murphy, B., and Ko, A.J. Characterizing the differences between pre- and post- release versions of software. In *ICSE '11: Proceedings of the 33rd International Conference on Software Engineering* (2011), 716--725.

[42] Liblit, B., Begel, A., and Sweetser, E. Cognitive Perspectives on the Role of Naming in Computer Programs. In *Proceedings of the 18th Annual Psychology of Programming Interest Group Workshop* (2006).

[43] Mockus, A. Organizational volatility and its effects on software defects. In *FSE '10: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2010), 117--126.

[44] Allamanis, M. and Sutton, C. Mining source code repositories at massive scale using language modeling. In *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories* (2013), 207--216.

[45] Hewner, M. Undergraduate conceptions of the field of computer science. In *ICER '13: Proceedings of the international ACM conference on International computing education research* (2013), 107--114.

[46] Sudol, L.A. and Jaspan, C. Analyzing the strength of undergraduate misconceptions about software engineering. In *ICER '10: Proceedings of the international workshop on Computing education research* (2010), 31--40.

[47] Begel, A. and Simon, B. Novice software developers, all over again. In *ICER '08: Proceedings of the International Workshop on Computing Education Research* (2008), 3--14.

[48] Dagenais, B., Ossher, H., Bellamy, R.K.E., Robillard, M.P., and Vries, J.P.d. Moving into a new software project landscape. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (2010), 275--284.

[49] Radermacher, A. and Walia, G. Gaps between industry expectations and the abilities of graduates. In *SIGCSE '13: Proceeding of the 44th ACM technical symposium on Computer science education* (2013), 525--530.

[50] Brechner, E. Things they would not teach me of in college: what Microsoft developers learn later. In *OOPSLA '03: Companion of the ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (2003), 134--136.

[51] Abelson, H., Sussman, G.J., and Sussman, J. *Structure and interpretation of computer programs*. MIT Press, 1985.

[52] Punter, T., Ciolkowski, M., Freimut, B.G., and John, I. Conducting on-line surveys in software engineering. In *ISESE '03: Proceedings of the International Symposium on Empirical Software Engineering* (2003), 80-88.